リセット機能を持つ順序回路に対するテスト系列圧縮法

樋上喜信[†] 高松雄三[†] 樹下行三[††]

本論文では，リセット機能を持つ順序回路のテスト系列圧縮法について提案する．提案するテスト系列圧縮法は静的圧縮法であり，与えられたテスト系列に対して，元の故障検出率を低下させないように，テスト系列の一部をリセット入力に置き換え，テスト系列を圧縮する．まず，与えられたテスト系列に対して故障シミュレーションを行い，置換の候補となるテストベクトルを選択する．次に，候補のテストベクトルの一部をリセット入力と置換する．その際，故障検出率が低下しないように，1) リセット状態とテストベクトルの印加の際に必要な状態との比較，2) 論理シミュレーションによる状態遷移の計算，3) 故障シミュレーションによる検出故障の計算，などを行う．提案法をプログラム化し，ベンチマーク回路に対して行った実験の結果によって，その有効性を確認した．

Test Sequence Compaction Method for Sequential Circuits with Reset States

Yoshinobu Higami,[†] Yuzo Takamatsu[†] and Kozo Kinoshita[††]

In this paper, we propose a static test sequence compaction method for sequential circuits with reset states under single stuck-at fault model. The proposed method first finds unremovable vectors by fault-dropping fault simulation or by non-fault-dropping fault simulation. Next, a subset of test vectors other than unremovable vectors are replaced with a reset signal. Detection of faults that are detected by an original test sequence is guaranteed by logic simulation and fault simulation for test subsequences. Experimental results for benchmark circuits show the effectiveness of the proposed method.

1. はじめに

順序回路に対するテスト生成は大変困難な問題として知られている．そこで，この問題の困難さを緩和するために，様々なテスト容易化の手法が用いられている．なかでもとくに，スキャン設計とよばれる手法が広く用いられており，これによって高い故障検出率を達成するテスト系列を，容易に生成できるようになる．しかしながら，スキャン設計では，回路面積の増大や，回路の性能の低下などの欠点があり，これを避ける手法として，リセット機能を付加する手法がある[8]．リセット機能を付加することによって，テスト生成における状態の正当化が容易になり，高い故障検出率が達成される．

テスト生成において，故障検出率とともに重要な要素として，テスト系列長がある．テスト系列長は，テスト時間や，テスタに必要なメモリ量に影響するため，できるだけ短い系列長が求められている．テスト系列長を短縮するテスト系列圧縮法には，これまで多くの手法が提案されており，それらは主に動的圧縮法と静的圧縮法に分けられる．動的圧縮法とは，テスト生成の途中においてテスト系列圧縮を行う手法である．過去の文献 5)，12)，14)，15) では，値の割り当てられていない外部入力に対して適当な値を割り当て，より多くの故障を検出するような手法がいくつか提案されている．たとえば，遺伝的アルゴリズムを用いる手法[14),15)]，ランダムに値を割り当てる手法[5)]，テスト生成法を応用する手法[12)]，などが提案されている．また，静的圧縮法を応用した動的圧縮法も提案されている[9)]．

静的圧縮法とは，テスト生成後にテスト系列圧縮を行う手法であり，テスト生成法に独立であるため，どのようなテスト系列に対しても適用可能であるという利点を持つ．これまで多くの手法が提案されており，テストベクトルを重ね合わせる手法[13)]，テストベクトルを省略・挿入・選択する手法[10)]，テストベクトルを削除・復帰する手法[2)]，状態遷移を考慮してテストベクトルを削除する手法[3),4)]，遺伝的アルゴリズムによ

† 愛媛大学工学部情報工学科
Faculty of Engineering, Ehime University
†† 大阪学院大学情報学部
Faculty of Informatics, Osaka Gakuin University

り部分系列の印加順序を決定する手法[1)]，などが提案されている．

本論文では，リセット機能を持つ順序回路に対するテスト系列の静的圧縮法を提案する．対象故障は単一縮退故障で，リセット状態はただ1つであり，テスト中は複数回リセット可能な環境を仮定する[11)]．リセット機能を持つ順序回路のテスト生成法は過去にも提案されており，そのような手法を基に動的圧縮法を適用することも効果的と考えられる．しかしながら，動的圧縮法はテスト生成法に依存するため汎用性に欠け，また，動的圧縮を行った後でさえ，静的圧縮を行うことでさらにテスト系列を短縮することができると考えられる．このような理由により，本論文では静的圧縮法の適用を考える．

提案法は2つあり，各手法とも以下の操作を行いテスト系列圧縮を実現する．

1) テストベクトルを除去可能ベクトルと除去不可能ベクトルに分類する．
2) 除去可能ベクトルの一部をリセット入力に置換する．

2つの手法は，それぞれ手法1，手法2とよび，除去可能ベクトルの定義やリセット入力に置換する手順が異なる．手法2は計算量が多い反面，手法1より多くの除去可能ベクトルを得ることができるため，最終的により短いテスト系列を得ることが期待できる．

本論文の以下の構成は次のようである．2章では手法1について，3章では手法2について説明する．4章では手法1，手法2を用いてISCAS'89ベンチマーク回路に対して行った実験の結果を示す．5章で本論文のまとめを述べる．

2. 手 法 1

2.1 概 要

手法1は次のような手順からなる．

1) テストベクトルを除去可能ベクトルと除去不可能ベクトルに分類する．
2) 除去可能ベクトルの次状態とリセット状態を比較し，リセット入力で置換できる除去可能ベクトルを見つけ，置換する．
3) 論理シミュレーションを行いリセット入力で置換できる除去可能ベクトルを見つけ，置換する．
4) 故障シミュレーションを行いリセット入力で置換できる除去可能ベクトルを見つけ，置換する．

2.2 テストベクトルの分類

テストベクトルの分類には故障シミュレーションで得られた情報を用いる．ここでの故障シミュレーションは，故障が検出された時点で故障をリストから削除するような，一般的な故障シミュレーションである．

まず，ある故障 f に対して，**故障検出ベクトル**と**故障伝搬ベクトル**を定義する．

［定義1］故障 f について，故障の影響が外部出力に伝搬するときの入力ベクトルを，故障 f の故障検出ベクトルとよぶ．

［定義2］故障 f について，時刻 i で故障の影響がFF（フリップフロップ）に伝搬し，時刻 $i+1$ の入力ベクトルが故障検出ベクトルまたは故障伝搬ベクトルであるとき，時刻 i の入力ベクトルを故障 f の故障伝搬ベクトルとよぶ．

［**テストベクトル分類法1**］

与えられたテスト系列で検出される故障の集合を $F=\{f_1,f_2,\ldots,f_m\}$ とする．故障 f_j $(j=1,2,\ldots,m)$ に対して，初めて検出される時刻の故障検出ベクトルと，それより前の時刻の故障伝搬ベクトルを含む集合 T_j を求める．

与えられたテスト集合を T とするとき，除去不可能ベクトルの集合 T_{un} と除去可能ベクトルの集合 T_{rm} を

$$T_{un}=\bigcup_{j=1}^{m}T_j$$

$$T_{rm}=T-T_{un}$$

として求める．

［**テストベクトル分類の例**］

テスト系列 $t_1,t_2,t_3,\ldots,t_7$ が与えられ，故障 f_1，f_2 に対する故障の影響の伝搬が，**表1**のようであったとする．ここで，'F'は故障の影響がFFに，'D'は故障の影響が外部出力に伝搬したことを表す．このとき，故障 f_1 に対しては，t_5，t_6 が故障伝搬ベクトルで，t_7 が故障検出ベクトルとなる．また，故障 f_2 に対しては，t_4，t_5 が故障伝搬ベクトルで，t_6 が故障検出ベクトルとなるが，t_2 は故障伝搬ベクトルとはならない．以上の結果より，t_1，t_2，t_3 が除去可能ベクトル，t_4，t_5，t_6，t_7 が除去不可能ベクトルとなる．

2.3 状態の比較

手法1では，テストベクトルを分類した後，まず除去可能ベクトルの次状態とリセット状態を比較し，そ

表1 故障伝搬の例
Table 1 Example of fault propagation.

ベクトル	t_1	t_2	t_3	t_4	t_5	t_6	t_7
故障 f_1					F	F	D
故障 f_2		F		F	F	D	

F：故障の影響がFFに伝搬
D：故障の影響が外部出力に伝搬

Algorithm: *Replace removable vectors with reset signals by logic simulation*
/* $t_1, t_2, \ldots, t_n$: a given test sequence; s_i : present state for t_i */
/* RS : a reset signal; $T_{compact}$: the obtained compacted sequence */
/* T_{sub} : target test subsequence for logic simulation */
/* + : concatenation of test vectors */
$T_{compact} = T_{sub} = \phi$;
for $(i = n\ ;\ i \geq 1\ ;\ i--)$
  if (t_i is an unremovable vector)
    $s_{compare} = s_i$;
    $T_{compact} = t_i + T_{sub} + T_{compact}$;
    $T_{sub} = \phi$;
  else if $(T_{sub} = \phi)$
    $T_{sub} = t_i$;
  else
    Perform logic simulation with $RS + T_{sub}$;
    if (The final state is equivalent to $s_{compare}$)
      $T_{compact} = RS + T_{sub} + T_{compact}$;
      $T_{sub} = \phi$;
      while(t_{i-1} is a removable vector && $i > 1$)
        $i--$;
    else
      $T_{sub} = t_i + T_{sub}$;

図 1 論理シミュレーションによるテストベクトルの置換アルゴリズム
Fig. 1 Algorithm to replace removable vectors with reset signals by logic simulation.

れらが一致するとき，除去可能ベクトルをリセット入力と置換する．たとえば，表 1 の例において，もし，t_3 の次状態がリセット状態と一致するなら，t_1，t_2，t_3 をリセット入力と置換する．テストベクトル t_4，t_5，t_6，t_7 に対しては，現状態が変化しないので，元と同じ故障が必ず検出される．

2.4 論理シミュレーションによる置換

次状態がリセット状態ではない除去可能ベクトルをリセット入力と置換した場合，回路の状態遷移が変化し，元と同じ故障が検出されない場合が生じる．しかしながら，手法 1 のテストベクトルの分類では，除去不可能ベクトルに対する現状態に変化がなければ，元と同じ故障の検出が保証される．そこで，論理シミュレーションを行うことによって，除去不可能ベクトルに対する現状態の変化を調べる．たとえば，ある除去可能ベクトルをリセット入力に置換したとする．もしそれ以降にある除去不可能ベクトルに対する現状態に変化がなければ，そのような置換をしても，元と同じ故障が検出されることが保証される．

図 1 に論理シミュレーションによるテストベクトルの置換アルゴリズムを示す．図中において，$t_1, t_2, \ldots, t_n$ は与えられたテスト系列を，t_i $(i = 1, \ldots, n)$ はテストベクトルを，s_i はテストベクトル t_i に対する現状態を，RS はリセット入力を，$T_{compact}$ は圧縮されたテスト系列を，T_{sub} は論理シミュレーションの対象となる部分系列を，+ は部分系列の結合を，それぞれ表す．

［**論理シミュレーションによるテストベクトル置換の例**］
テスト系列 $t_1, t_2, t_3, \ldots, t_7$ が与えられ，t_1，t_6，t_7 を除去不可能ベクトル，t_2，t_3，t_4，t_5 を除去可能ベクトルとする．また，先の「状態の比較」において，t_2，t_3，t_4，t_5 の次状態はリセット状態と異なる，ということが分かっているとする．まず t_5 印加時の現状態をリセット状態にできるか調べる．具体的には，回路の状態をリセット状態にし，テストベクトル t_5 を印加し，その後の回路の状態を調べる．もし，それが t_6 に対する現状態と一致するなら，t_2，t_3，t_4 をリセット入力と置換する．もし，一致しない場合には，t_4 印加時の現状態をリセット状態にできるか調べる．さらにそれもできない場合には，t_3，t_2 の順に，同様のことを調べる．

2.5 故障シミュレーションによる置換

除去不可能ベクトルに対する現状態が変化する場合であっても，元と同じ故障が検出されることが考えられる．そこで，ある除去可能ベクトルをリセット入力で置換したときに，元と同じ故障が検出されるかどうかを，故障シミュレーションによって調べる．この場合の故障シミュレーションの対象となるテストベクトルと故障は，一部のテストベクトルと，それらによって検出されるべき故障のみであるため，テスト系列全

Algorithm: *Replace removable vectors with reset signals by fault simulation*
/* $t_1, t_2, \ldots, t_n$: a given test sequence */
/* RS : a reset signal , $T_{compact}$: the obtained compacted sequence */
/* F_{target} : target faults for fault simulation; T_{sub} : target test subsequence for fault simulation */
/* + : concatenation of test vectors */
$F_{target} = T_{compact} = T_{sub} = \phi$;
for $(i = n; i \geq 1; i--)$
  if (t_i is the first fault-detecting vector)
    $F_{target} = F_{target} \cup \{f{:}t_i$ propagates the effect of f to some primary outputs for the first time$\}$;
    $T_{sub} = t_i + T_{sub}$;
  else if (t_i is a fault-propagating vector)
    $T_{sub} = t_i + T_{sub}$;
  else if ($T_{sub} \neq \phi$)
    Perform fault simulation with $RS + T_{sub}$;
    if (All the faults in F_{target} are detected)
      $T_{compact} = RS + T_{sub} + T_{compact}$;
      $F_{target} = T_{sub} = \phi$;
    else
      $T_{sub} = t_i + T_{sub}$;

図 2 故障シミュレーションによるテストベクトルの置換アルゴリズム
Fig. 2 Algorithm to replace removable vectors with reset signals by fault simulation.

表 2 テスト系列の例
Table 2 Example of a test sequence.

vector	t_1	t_2	t_3	t_4	t_5	t_6	t_7
detected fault	f_1		f_2			f_3	f_4

体を対象とした故障シミュレーションと比較して，計算量は少ない．

図 2 に故障シミュレーションによるテストベクトルの置換アルゴリズムを示す．図中において，$t_1, t_2, \ldots, t_n$，RS，$T_{compact}$，+ は図 1 と同様の意味である．F_{target}，T_{sub} は，故障シミュレーションの対象となる故障集合と部分系列を表す．

［**故障シミュレーションによるテストベクトル置換の例**］
表 2 に示されるようなテスト系列が与えられ，故障 f_1，f_2，f_3，f_4 がそれぞれ t_1，t_3，t_6，t_7 によって初めて検出されるとする．まず，リセット状態で t_6，t_7 を印加して，故障 f_3，f_4 が検出されるかどうかを調べる．もし検出されれば，テストベクトル t_4，t_5 をリセット入力と置換する．もしそうでない場合には，リセット状態で t_5，t_6，t_7 を印加して，故障シミュレーションを行う．このときも対象故障は f_3，f_4 のみである．もし t_4，t_5 のいずれもリセット入力と置換できないなら，次にリセット状態で t_3，t_4，t_5，t_6，t_7 を印加して，故障シミュレーションを行う．このときの対象故障は，f_2，f_3，f_4 である．もしこれらの故障がすべて検出されたなら，t_2 をリセット入力で置換する．

2.6 手法 1 によるテスト系列圧縮の例

これまでに説明した手法 1 の各手順に従ったテスト系列の圧縮について，例を用いて説明する．テスト系列 $t_1, t_2, \ldots, t_{13}, t_{14}$ と，各テストベクトルに対する次状態，各テストベクトルで初めて検出される故障が，それぞれ**表 3** に示されるように与えられたとする．また，リセット状態が状態 A と同一であると仮定する．まず，テストベクトルを除去可能ベクトルと除去不可能ベクトルに分類する．ここでは，表 3 に示されるように，t_3，t_4，t_7，t_8，t_9，t_{11}，t_{12} が除去可能ベクトル，それ以外が除去不可能ベクトル，のように分類されたとする．次に，リセット状態と各テストベクトルの次状態を比較する．その結果，t_4 印加後の状態がリセット状態と一致するので，除去可能ベクトル t_3，t_4 をリセット入力と置換する．次に論理シミュレーションを行い，除去可能ベクトルをリセット入力と置換する．1 回目の論理シミュレーションでは，リセット状態から t_{12} を印加し，終状態が E と一致するかどうかを調べる．一致しない場合，次の論理シミュレーションでは，リセット状態から t_9 を印加し，終状態が B と一致するかどうかを調べる．もし，終状態が B と一致したなら，テストベクトル t_7，t_8 をリセット入力で置換する．論理シミュレーションによる置換が終了した後，故障シミュレーションによる置換を実行する．まず最初の故障シミュレーションでは，リセット状態から系列 t_{13}，t_{14} を印加し，故障 f_7，f_8 が検出されるかどうかを調べる．もし，故障 f_7，f_8 が検出された場合，テストベクトル t_{11}，t_{12} をリセット入力と置換する．以上のような置換によって，得られたテスト系列は，$t_1, t_2, RS, t_5, t_6, RS, t_9, t_{10}, RS, t_{13}, t_{14}$ のよ

表 3 テスト系列の例
Table 3 Example of a test sequence.

ベクトル	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}
分類	UN	UN	R	R	UN	UN	R	R	R	UN	R	R	UN	UN
次状態	B	D	C	A	B	E	C	D	B	D	C	E	B	E
検出故障	f_1	f_2			f_3	f_4				f_5, f_6			f_7	f_8

R：除去可能ベクトル UN：除去不可能ベクトル

うになる．ここで RS はリセット入力を表す．

3. 手 法 2

3.1 概 要

手法 2 では，除去可能ベクトルと除去不可能ベクトルの定義が手法 1 と異なっており，また手法 1 とは異なった計算法によってそれらの分類を行う．手法 2 では，除去可能ベクトルができるだけ多くなるように分類する．除去可能ベクトルをすべて除去できるわけではないが，除去できる候補数が増えれば，より多くのテストベクトルをリセット入力に置換でき，結果としてより短いテスト系列を得ることが期待できる．ただし，多くの除去可能ベクトルを発見するためには，多くの計算量を必要とし，また，除去可能ベクトルをリセット入力と置換する場合にも，故障シミュレーションによって，元の故障の検出を保証しなければならない．

手法 2 は次のような手順からなる．

1) テストベクトルを除去可能ベクトルと除去不可能ベクトルに分類する．
2) 故障シミュレーションを行いリセット入力で置換できる除去可能ベクトルを見つけ，除去する（手法 1 と同様の計算）．

3.2 テストベクトルの分類

ここでは，与えられたテスト系列に対して，ノンドロップ故障シミュレーションを行う．ノンドロップ故障シミュレーションとは，故障が検出されても，故障を故障リストから削除せずに故障シミュレーションを行う手法で，故障が検出されるテストベクトルを複数記憶する．ただしここでは，故障を検出するテストベクトルを記憶する最大数に上限を設けることで，シミュレーション時間と記憶容量の増大を防ぐ．

まずノンドロップ故障シミュレーションを行い，各故障に対して複数個の故障検出ベクトルを求める．次に，元のテスト系列で検出されるすべての故障に対して，各テストベクトルが故障検出ベクトルであるかどうかを要素とする行列を生成する．たとえば，故障 f_i に対し，テストベクトル t_j が故障検出ベクトルであるとき，i 行 j 列の要素を 1，そうでない場合は 0 のように表す．得られた行列を基に，被覆問題を解くことで，すべての故障に対して，少なくとも 1 つの故障検出ベクトルが含まれるような，最小のテストベクトルの集合を得ることを目指す．被覆問題を解くアルゴリズムとして，本研究では貪欲アルゴリズムを用いる．結果として得られたテストベクトルを除去不可能ベクトル，その他を除去可能ベクトルとして，以降の除去可能ベクトルのリセット入力による置換操作に進む．

表 4 故障検出ベクトルの例
Table 4 Example of fault detecting vectors.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7
f_1	0	1	0	0	1	0	0
f_2	0	0	1	0	0	0	1
f_3	1	0	1	0	0	1	0
f_4	0	0	0	1	1	0	0
f_5	1	0	1	0	0	0	1

'1'：故障 f_i に対する故障検出ベクトル

［**テストベクトル分類の例**］

テストベクトル $t_1, \ldots, t_7$ と故障 $f_1, \ldots, f_5$ が与えられ，故障シミュレーションの結果，**表 4** に示すように，故障検出ベクトルが求められたとする．被覆問題を解くことで，すべての故障に対して，少なくとも 1 つの故障検出ベクトルを含むような最小のテストベクトルの集合として，$\{t_3, t_5\}$ が選ばれる．したがって，t_3，t_5 が除去不可能ベクトル，それ以外が除去可能ベクトルとなる．

4. 実 験 結 果

提案法を C 言語を用いてプログラム化し，計算機（PentiumIII 600 MHz）上において ISCAS'89 ベンチマーク回路に対して行った実験の結果を示す．**表 5** は，元の系列として，HITEC アルゴリズム[7]を用いて縮退故障に対して生成されたテスト系列を用いた結果を示す．実験では，すべての FF が 0 となる状態をリセット状態とし，手法 2 における 1 つの故障の検出回数の上限は 10 とした．表中には，元のテスト系列（original の列），手法 1 によって得られたテスト系列長（length の下の m1 の列）と元の系列長に対する割合（length の下の m1 の右の%の列），手法 2 によって得られたテスト系列長（length の下の m2 の列）と元の系列長に対する割合（length の下の m2 の右の%の列），リセット入力の印加回数（reset の下の m1，m2 の列），計算時間（cputime の下の m1，m2 の列）がそれぞれ示されている．実験の結果，手法 1 では平均

表 5 HITEC テスト系列に対する実験結果
Table 5 Experimental results for test sequences generated by HITEC.

	length					reset		cputim(s)	
ckt	original	m1	%	m2	%	m1	m2	m1	m2
s344	127	94	74.0	83	65.4	9	7	0.02	0.06
s349	134	98	73.1	79	59.0	12	9	0.02	0.06
s382	2074	1540	74.3	1307	63.0	9	11	4.64	6.37
s386	286	225	78.7	185	64.7	34	34	0.06	0.12
s400	2214	1517	68.5	1210	54.7	10	14	4.48	3.96
s420	166	152	91.6	140	84.3	9	12	0.09	0.15
s444	2240	1259	56.2	1183	52.8	11	14	5.01	4.56
s526	2258	1753	77.6	1322	58.5	4	11	2.82	11.33
s641	209	197	94.3	151	72.2	8	13	0.11	0.18
s713	173	165	95.4	137	79.2	9	13	0.08	0.17
s820	1115	1014	90.9	872	78.2	78	72	0.80	1.44
s832	1137	1051	92.4	890	78.3	70	72	1.06	1.65
s1196	435	411	94.5	340	78.2	25	55	0.42	1.03
s1238	475	456	96.0	364	76.6	20	59	0.64	1.29
s1423	150	150	100	131	87.3	1	5	0.47	1.21
s1488	1170	1107	94.6	943	80.6	61	59	2.06	3.24
s1494	1245	1160	93.2	1008	81.0	69	58	2.25	3.62
s5378	912	779	85.4	625	68.5	15	35	26.89	75.98
s35932	496	456	91.9	368	74.2	14	11	966.4	2250
average	851.5	679.8	85.4	567.4	71.8	23.6	28.3	50.9	118.3

m1：手法 1 m2：手法 2

表 6 圧縮した系列の故障検出率（%）
Table 6 Fault coverage by compacted test sequences.

ckt	original	method1	method2
s344	95.91	98.25	98.25
s349	95.43	97.14	97.14
s382	78.19	86.22	86.22
s386	81.77	81.77	81.77
s400	82.55	85.14	83.02
s420	41.63	47.21	47.21
s444	82.07	83.54	82.49
s526	65.05	77.48	77.84
s641	86.51	87.37	87.37
s713	81.93	82.62	82.62
s820	95.65	95.77	95.77
s832	93.91	94.02	94.02
s1196	99.76	99.76	99.76
s1238	94.69	94.69	94.69
s1423	47.72	49.11	48.98
s1488	97.17	97.31	97.31
s1494	96.48	96.61	96.61
s5378	70.19	71.78	72.39
s35932	89.28	89.30	89.30

で元の約 85%（average の段の左から 4 番目）に，手法 2 では平均で元の約 72%（average の段の左から 6 番目）の系列長に短縮された．また，すべての回路で手法 2 の方がより短くなった．計算時間に関しては，手法 2 の平均（average の段の右から 1 番目）は手法 1 の平均（average の段の右から 2 番目）の約 2 倍程度であった．提案法では，故障検出率を少しでも増加させるため，テスト系列を印加する前にリセットを行うと仮定した．そのため s1423 に対する手法 1 の結果では，系列長に変わりはないが，リセット回数が 1 回となっている．

表 6 に，元のテスト系列（original の列）と手法 1，手法 2 によって得られたテスト系列による故障検出率（method1 と method2 の列）を示す．表 6 より，提案法の圧縮によって，若干であるが元の故障検出率より高い故障検出率が得られていることが分かる．これは，テストベクトル数は減少したが，回路の状態遷移が変化したため，それまで未検出であった故障が，新たに検出されるようになったためと考えられる．

HITEC で生成されたテスト系列を逆順復帰法（Reverse Order Restoration）[2]によって圧縮したテスト系列に対して，提案手法を適用した実験の結果を**表 7** に示す．逆順復帰法[2]とは，リセット状態を用いない静的圧縮法であり，与えられたテスト系列から初期化系列を除いてすべて削除した後，故障シミュレーションによって削除したテストベクトルの一部を復帰させる手法である．表 7 には，左から順に，回路名（ckt の列），HITEC で生成された系列長（original の列），逆順復帰法により得られた系列長（ROR の列），逆順復帰法を適用して得られたテスト系列を手法 1 によって圧縮した系列長（method1 の length の列）とリセット回数（method1 の reset の列），逆順復帰法を適用して得られたテスト系列を手法 2 によって圧縮した系列長（method2 の length の列）とリセット回数（method2 の reset の列）を示す．実験の結果，逆順復帰法で十分に圧縮されたテスト系列であっても，提案法を用いることによって，さらに圧縮できることが示された．ただし，s420 や s35932 のようにほとんど変化のない場合もあった．

表 7 に示された結果を他の圧縮法による結果と比較した．**表 8** には，HITEC による系列長（original の列），逆順復帰法による系列長（ROR の列），逆順復帰法を適用して得られたテスト系列を提案法の手法 2 によって圧縮した系列長（method2 の列），その系列長とリセット回数の和（method2+の列），Lin らによるテスト圧縮の結果[6]の系列長（Lin, et al. の列）を示す．ただし，s349，s386，s420 についての結果は，文献 6) に掲載されていないため比較を行っていない．

表 7 ROR テスト系列に対する実験結果
Table 7 Experimental results for test sequences generated by ROR.

ckt	original	ROR	method1 length	method1 reset	method2 length	method2 reset
s344	127	48	43	2	37	4
s349	134	49	48	2	44	5
s382	2074	430	384	3	331	4
s386	286	143	138	6	125	9
s400	2214	584	501	2	500	3
s420	166	93	93	1	93	1
s444	2240	418	277	4	276	5
s526	2258	611	610	2	605	3
s641	209	109	109	1	103	2
s713	173	93	93	1	87	2
s820	1115	597	594	4	592	6
s832	1137	605	594	7	588	9
s1196	435	252	251	2	247	5
s1238	475	267	266	2	253	11
s1423	150	114	114	1	111	2
s1488	1170	643	638	6	636	7
s1494	1245	614	608	7	606	9
s5378	912	469	468	2	417	14
s35932	496	149	149	1	148	2

表 8 他手法との実験結果の比較
Table 8 Comparison of experimental results with other results.

ckt	original	ROR	method2	method2+	Lin, et al.
s344	127	48	37	41	50
s382	2074	430	331	335	353
s400	2214	584	500	503	584
s444	2240	418	276	281	480
s526	2258	611	605	608	649
s641	209	109	103	105	112
s713	173	93	87	89	93
s820	1115	597	592	598	599
s832	1137	605	588	597	597
s1196	435	252	247	252	256
s1238	475	267	253	264	272
s1423	150	114	111	113	160
s1488	1170	643	636	643	613
s1494	1245	614	606	615	640
s5378	912	469	417	431	456
s35932	496	149	148	150	183

表 9 異なる検出回数の制限による比較
Table 9 Experimental results with different upper bounds of detection times.

ckt	length original	length max=10	length max=100	length %	cputime(s) max=10	cputime(s) max=100	cputime(s) %
s344	127	83	65	78.3	0.06	0.11	183.3
s349	134	79	61	77.2	0.06	0.10	166.7
s382	2074	1307	1284	98.2	6.37	4.92	77.2
s386	286	185	170	91.9	0.12	0.18	150.0
s400	2214	1210	1065	88.0	3.96	4.25	107.3
s420	166	140	135	96.4	0.15	0.17	113.3
s444	2240	1183	1175	99.3	4.56	4.88	107.0
s526	2258	1322	1259	95.2	11.33	12.89	113.8
s641	209	151	146	96.7	0.18	0.26	144.4
s713	173	137	128	93.4	0.17	0.27	158.8
s820	1115	872	863	99.0	1.44	2.02	140.3
s832	1137	890	876	98.4	1.65	2.17	131.5
s1196	435	340	340	100	1.03	1.35	131.1
s1238	475	364	350	96.2	1.29	1.59	123.3
s1423	150	131	131	100	1.21	1.39	114.9
s1488	1170	943	895	94.9	3.24	4.86	150.0
s1494	1245	1008	996	98.8	3.62	5.45	150.6
s5378	912	625	708	113.3	75.98	90.23	118.8

Lin らの手法は，逆順復帰法[2)]をさらに改良した手法であり，元の系列として HITEC による同じ系列長のテスト系列を用いた結果である．表 8 より，s1488 を除くすべての回路について，Lin らの手法より，提案する手法 2 によるテスト系列長が短くなった．

手法 2 のテストベクトルの分類の際の故障シミュレーションにおける，1 つの故障の検出回数の上限を 10，100 のように変化させたときの結果を**表 9** に示す．表 9 には，元の系列長（original の列），上限を 10，100 にした場合の圧縮された系列長（length の下の max=10，max=100 の列），上限 100 の系列長の上限 10 の系列長に対する割合（length の下の%の列），

上限を 10，100 にした場合の計算時間（cputime の下の max=10，max=100 の列），上限 100 の計算時間の上限 10 のそれに対する割合（cputime の下の%の列）を示す．実験の結果，一部の回路を除くほとんどの回路において，上限を 100 にした方が，短い系列が得られている．しかし，s5378 では，上限を 10 にした方が短い系列が得られている．一般には，上限を大きくした方が，故障検出ベクトルが増加し，選択肢が増えるため除去可能ベクトルが多くなると考えられる．しかし，除去可能ベクトル選択のための被覆問題を解く際に，貪欲アルゴリズムを用いているため，必ずしも最小の解が得られず，結果として，除去可能ベクトルが減少する場合があり，得られた系列長も増大したと考えられる．また，貪欲アルゴリズムにより得られた近似的最適解が必ずしもテスト系列圧縮の最適解とはならない可能性等が考えられる．計算時間に関しては，上限を 100 にした場合には，約 7〜83%計算時間が増加している．しかし，s382 では，約 33%計算時間が減少している．これは，上限が 10 の場合に，置換するベクトルを求めるための故障シミュレーションに要する時間が長かったためである．

5. おわりに

本論文では，リセット機能を持つ順序回路に対するテスト系列の静的圧縮法を提案した．提案法は 2 つあり，手法 1，手法 2 とも，テストベクトルの分類と，その結果に基づく，テストベクトルのリセット入力による置換を行った．テストベクトルの分類では，故障シミュレーションで得られた情報を用い，除去可能ベクトルと除去不可能ベクトルに分類した．手法 1 では，状態の比較，論理シミュレーションや故障シミュレーションを用いた，除去可能ベクトルのリセット入力による置換を行った．また，手法 2 では，故障シミュレーションを用いた，除去可能ベクトルのリセット入力による置換を行った．手法 2 は，ノンドロップ故障シミュレーションによって，手法 1 より多い除去可能ベクトルを得た．その結果，多くの除去可能ベクトルがリセット入力で置換され，最終的により短いテスト系列を得ることができた．反面，手法 2 は手法 1 に比べて，平均で約 2 倍の計算時間を要した．また，元の系列を逆順復帰法という静的圧縮法で圧縮したテスト系列に対して提案法を適用した結果，さらにテスト系列が圧縮された．このことより，提案法は，他のテスト圧縮法で圧縮されたテスト系列に対しても，有効であることが示された．

しかしながら，最短のテスト系列長を求めるためには，逆順復帰法以外のテスト系列圧縮法と提案手法の組合せや，それらの適用順序の最適化について今後検討する必要がある．また，一部の FF がリセット可能である部分リセット回路への応用などについても検討し，有効な手法を開発することが今後の課題である．

参考文献

1) Corno, F., Prinetto, P., Rebaudengo, M. and Reorda, M.S.: New Static Compaction Techniques of Test Sequences for Sequential Circuits, *Proc. European Design and Test Conf.*, pp.37–43 (Mar. 1997).
2) Guo, R., Pomeranz, I. and Reddy, S.M.: On Speed-Up Vector Restoration Based Static Compaction of Test Sequences for Sequential Circuits, *Proc. Asian Test Symp.*, pp.467–471 (Dec. 1998).
3) Hsiao, M.S., Rundnick, E.M. and Patel, J.H.: Fast Algorithms for Static Compaction of Sequential Circuit Test Vectors, *Proc. VLSI Test Symp.*, pp.188–195 (1997).
4) Hsiao, M.S., Rundnick, E.M. and Patel, J.H.: Fast Static Compaction Algorithms for Sequential Circuit Test Vectors, *IEEE Trans. Comput.*, Vol.48, pp.311–322 (1999).
5) Lambert, T.J. and Saluja, K.K.: Methods for Dynamic Test Vector Compaction in Sequential Test Generation, *Proc. VLSI Design Conf.*, pp.166–169 (Jan. 1996).
6) Lin, X., Cheng, W.-T., Pomeranz, I. and Reddy, S.M.: SIFAR: Static Test Compaction for Synchronous Sequential Circuits Based on Single Fault Restoration, *Proc. VLSI Test Symp.*, pp.205–212 (Apr. 2000).
7) Niermann, T.M. and Patel, J.H.: HITEC: A test generation package for sequential circuits, *Proc. European Conf. on Design Automation*, pp.214–218 (Feb. 1991).
8) Pomeranz, I. and Reddy, S.M.: On the Role of Hardware Reset in Synchronous Sequential Circuit Test Generation, *IEEE Trans. on Computers*, pp.1100–1105 (Sep. 1994).
9) Pomeranz, I. and Reddy, S.M.: Dynamic Test Compaction for Synchronous Sequential Circuits Using Static Compaction Techniques, *Proc. Int. Symp. on Fault-Tolerant Comp.*, pp.53–61 (June 1996).
10) Pomeranz, I. and Reddy, S.M.: On Static Compaction of Test Sequences for Synchronous Sequential Circuits, *Proc. Design Automation Conf.*, pp.215–220 (June 1996).
11) Pomeranz, I. and Reddy, S.M.: On the Use of Reset to Increase the Testablility of Intercon-

nected Finite-State Machine, *Proc. European Design and Test Conf.*, pp.554–559 (Mar.1997).
12) Raghunathan, A. and Chakradhar, S.T.: Acceralation Techniques for Dynamic Vector Compaction, *Dig. Int. Conf. on Computer-Aided Design*, pp.310–317 (Nov. 1996).
13) Roy, R.T., Niermann, T.M., Patel, J.H., Abraham, J.A. and Saleh, R.A.: Compaction of ATPG-Generated Test Sequences for Sequnetial Circuits, *Dig. Int. Conf. on Computer-Aided Design*, pp.382–385 (Nov. 1988).
14) Rundnick, E.M. and Patel, J.H.: Efficient Techniques for Dynamic Test Sequence Compaction, *IEEE Trans. Comput.*, Vol.48, pp.323–330 (1999).
15) Santosa, Y., Merten, M., Rundnick, E.M. and Abramovici, M.: FreezeFrame: Compact Test Generation Using a Frozen Clock Strategy, *Proc. Design Automation and Test in Europe*, pp.747–752 (Mar. 1999).

(平成 12 年 9 月 24 日受付)
(平成 13 年 2 月 1 日採録)

樋上 喜信（正会員）

平成 8 年大阪大学大学院工学研究科応用物理学専攻博士後期課程修了．同年日本学術振興会特別研究員採用．平成 10 年より愛媛大学工学部助手．論理回路に対するテスト生成およびテスト容易化設計に関する研究に従事．電子情報通信学会，IEEE 各会員．博士（工学）．

高松 雄三（正会員）

昭和 41 年愛媛大学工学部電気工学科卒業．佐賀大学理工学部電子工学科助教授を経て，昭和 62 年 10 月より愛媛大学工学部情報工学科教授，現在に至る．論理回路の診断・テストに関する研究に従事．工学博士．著書「論理設計入門」（共著，日新出版）など．電子情報通信学会正員，IEEE Senior Member.

樹下 行三（正会員）

昭和 34 年大阪大学工学部通信工学科卒業，昭和 39 年同大学院工学研究科通信工学専攻博士課程修了，工学博士．同年同大学工学部電子工学科助手，昭和 41 年同学科助教授，昭和 53 年広島大学総合科学部教授，平成元年 10 月大阪大学工学部教授．平成 10 年 4 月大阪大学大学院工学研究科教授，平成 12 年 3 月大阪大学名誉教授，大阪学院大学情報学部教授．論理回路およびメモリのテスト容易化設計，故障診断，テスト生成などの研究に従事．昭和 61 年〜63 年情報処理学会設計自動化研究会主査，昭和 62 年〜平成元年電子情報通信学会フォールトトレラントシステム研究会専門委員長，昭和 63 年 IEEE・FTCS–18 プログラム委員長，平成 4 年 IEEE 第 1 回アジアテストシンポジュウム実行委員長など．電子情報通信学会フェロー，IEEE フェロー．